Universitat de Barcelona

Fundamentals of Data Science Master's Thesis

Bioactivity-oriented de novo design of small molecules by conditional variational autoencoders

---

Author:
Alex Castrelo Cid

Supervisor:
Dr. Jordi Vitrià Marca

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamentals of Data Science

in the

Facultat de Matemàtiques i Informàtica

July 1, 2019

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Bioactivity-oriented de novo design of small molecules by conditional variational autoencoders

by Alex Castrelo Cid

Deep generative networks are an emerging technology in drug discovery. Our work is divided in two parts. In the first one, we built a variational autoencoder (VAE) that is able to learn the grammar of the molecules, represent them in a latent space, and generate new ones. In the second one, we built and trained a conditional variational autoencoder (CVAE) that is capable of generating new molecules based on desired properties. We will see in detail the architecture of both models and how they were trained.

The molecule properties were provided by the Chemical Checker (CC), a resource of processed, harmonised and integrated small-molecule bioactivity data. We will generate different molecules with different target properties, and we will check how close the properties of the generated molecules are from the target ones. These properties are called signatures.

At the end of the project we sample CC signatures with different similarity to the input molecule signatures, and we show that the signatures of the molecules generated this way resemble the sampled signatures, meaning that we can generate new random molecules based on desired properties.

Acknowledgements

This MSc thesis would not have been possible without the help and support of many people to whom I would like to dedicate the following words.

Firstly, thanks to my supervisor, Dr. Jordi Vitrià Marca, for sharing his knowledge and experience.

Secondly, I would like to thank all my labmates for making my stay in the laboratory amazing. In particular, thanks to Dr. Patrick Aloy and Dr. Miquel Duran-Frigola for leading me through the project and being so patient explaining me the necessary biological/chemical background to proceed in the project. Also, big thanks to Dr. Martino Bertoni for being my drug dealer, for providing me the drug-like molecules and their properties when I needed them.

Finally, all my sincere gratitude to Gisela, for her support in the most stressful days during the Master, and for the help in the English writing process.

Contents

Chapter 1

Introduction

1.1 Motivation

The chemical space remains mostly unexplored. This means that it is possible to discover molecules never found before. On the other hand, not all molecules are feasible. Actually, only a small proportion of the theoretically possible 10^{60} drug-like molecules are therapeutically relevant, fact that makes the drug discovery process very difficult [1, 2]. Therefore, it is necessary to learn the rules of chemistry by analysing existing chemical matter.

De novo small molecule generation is a re-emerging field thanks to deep learning. Despite deep learning is not new, the new powerful distributed GPU-based systems have given a second chance to the field.

Recently, at *Institut de Recerca Biomèdica de Barcelona* (IRB), they have assembled the Chemical Checker (CC) [3], a large repository of bioactivity data for small molecules. The CC contains enough information to learn systematically the determinants of a large spectrum of activities. Due to it is possible to learn these determinants, there is the opportunity to generate new molecules with the desired bioactivities.

There are two main reasons to use these determinants: We want to generate molecules with the desired bioactivities, and, since the chemical space is unattainable, we need to do an efficient exploration of it in order to optimise resources and accelerate the drug discover pipeline.

Although many attempts have been done in the field of de novo molecule generation [4, 5, 6, 7, 8, 9], we were very eager to try out the CC bioactivity data as input of a conditional variational autoencoder.

1.2 Problem statement

The main goal is to generate small de novo molecules with the desired properties, which, in our case, are the CC signatures (explained in Section 2.2).

Since we are going to work with a neural network, we have to find a molecular representation that can be understood by the network (SMILES, section 2.1).

We need to find and tune a good model that is able to generate new molecules. In order to do so, the model has to learn the "grammar" of the molecules. We need to check if the molecules we are generating are valid or not, and if so, a way to compare the similarity between the generated molecules and their respective input molecule.

As next step, we will have to expand the model so it can take into account the properties of the molecule. Once the model is trained, we should be able to generate new molecules that target specific properties. We also need a way to check if the generated molecules have properties that are similar to our target.

On top of that, we need to be able to generate molecules that are in the region of interest in an efficient way, this is, that the ratio of those "interesting" generated molecules with respect to the total amount of generated molecules is above a certain threshold.

Chapter 2

Background

2.1 SMILES

In order to be fed to the algorithm, the molecules have to be processed as a sequence of characters. This is the case for the Simplified Molecular-Input Line-Entry System (SMILES) representation, which has been used in several generative algorithms [10].

Each molecule can be represented in different ways using the SMILES notation, but we will use what is called the canonical version of the smiles, which is unique for every molecule (see Figure 2.1).

2.2 Signatures

Recently, the Structural Bioinformatics and Network Biology (SBNB) laboratory, led by Dr. Patrick Aloy at IRB Barcelona, developed the Chemical Checker (CC) [3]. The CC is a resource of processed, harmonised and integrated small-molecule bioactivity data. The resource divides data into five levels of increasing complexity, ranging from the chemical properties of the compounds to their clinical outcomes. In between, it considers targets, off-targets, perturbed biological networks and several cell-based assays such as gene expression, growth inhibition and morphological profiles.

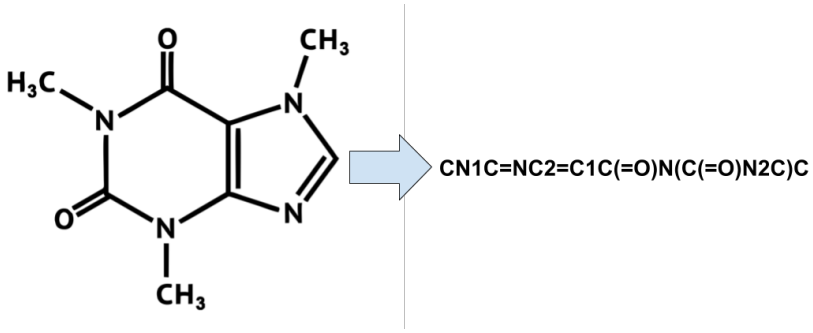


FIGURE 2.1: Caffeine molecule represented as a SMILES

The main assets of the CC are the so-called "bioactivity signatures". CC signatures are 128-dimensional vectors (embeddings) that encapsulate data of a certain kind by means of a two-step algorithm consisting, first, on the calculation of small-molecule similarities based on bioactivity data, followed by a network-embedding procedure performed on the resulting small-molecule similarity network. In this work, we capitalise on "binding" CC signatures, which embed physical interaction (binding) profiles of small molecules across a panel of >3,000 proteins.

We will use these signatures in the second model we implemented, and we will generate random molecules that would satisfy a particular signature as a constraint, i.e. molecules that would have a binding profile of interest.

2.3 Type of autoencoders

In this section we will present 3 different architectures of autoencoders, starting with the most simple one and introducing new changes that will adapt the network to our needs.

2.3.1 Autoencoder

Autoencoders (Figure 2.2) are a special type of neural network architecture in which the output is the same as the input. Autoencoders are trained in an unsupervised manner in order to learn low level representations of the input data (encoder). These low level features are then deformed back to project the actual data (decoder). An autoencoder is a regression task where the network is asked to predict its input (in other words, model the identity function). These networks have a tight bottleneck of a few neurons in the middle, forcing them to create effective representations that compress the input into a low-dimensional code that can be used by the decoder to reproduce the original input[1].

The encoded representation of the input is a dense vector or a fixed size which is called latent space. This latent space represents the lowest level space in which the input is reduced and information is preserved.

[1]Source: Kaggle

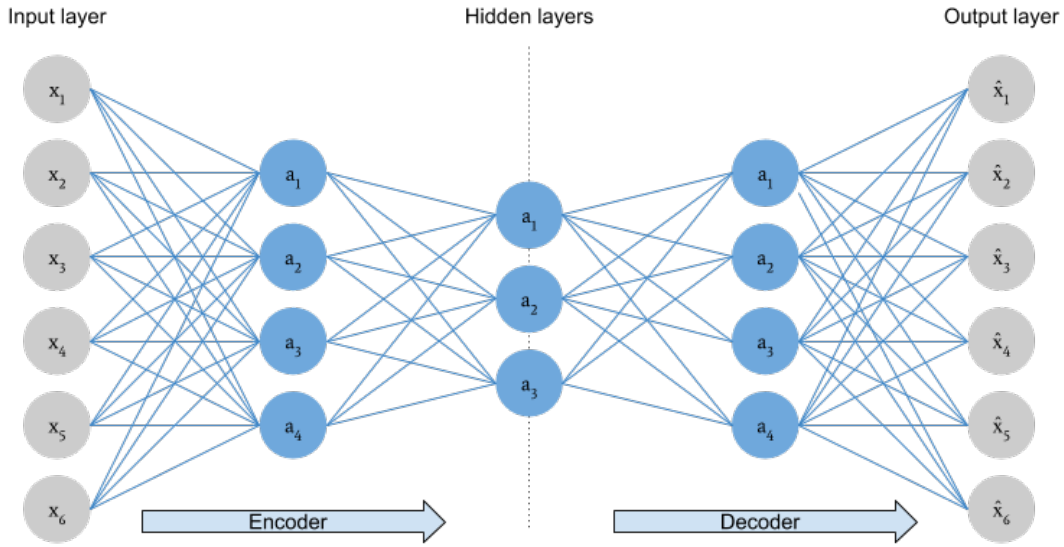


FIGURE 2.2: Structure of an autoencoder

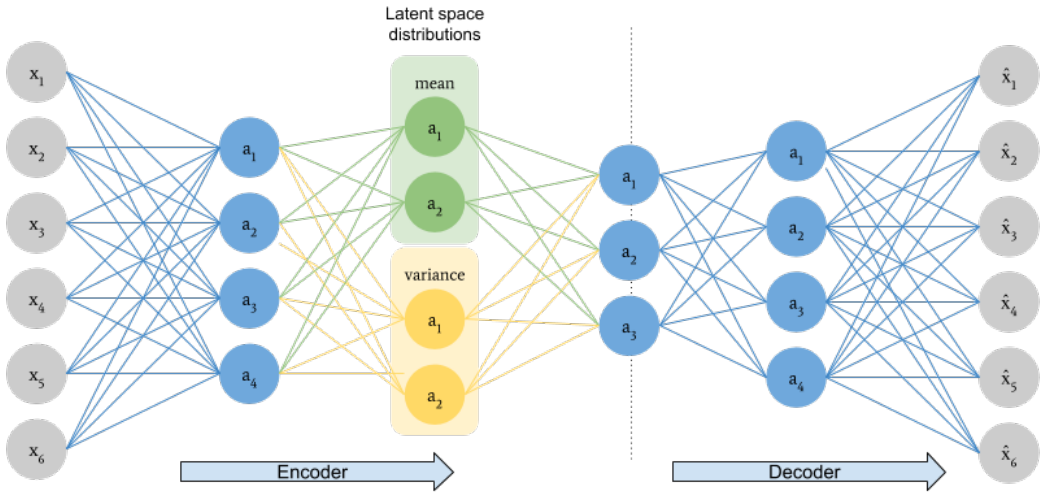


FIGURE 2.3: Structure of a variational autoencoder (VAE)

2.3.2 Variational autoencoder

On the other hand, autoencoders have a drawback: they cannot be used to generate new data. Introducing the variational autoencoder (VAE), a modified version of the regular autoencoder. Instead of just learning a function representing the data, they learn the parameters of a probability distribution (Gaussian for instance) representing the data. Since it learns to model the data, we can then sample from the distribution and generate new input data samples[2].

As it can be seen in figure 2.3, the structure of the VAE is very similar to the

[2]Source: Quora

regular autoencoder, with the difference that this time we have a couple of fully-connected layers prior to the latent layer. Those two layers are learning the probability distribution of this latent space, this is, the mean μ and variance σ.

In order to be able to keep the Gaussian shape of the latent space we have to penalise the network for creating latent spaces that are far from that distribution. This is done by just adding another term to the current loss function. This term is the so called Kullback-Leibler divergence (KL divergence). The KL divergence (also called relative entropy) is a measure of how one probability distribution is different from a second, reference probability distribution[3].

Essentially, what we are looking at with the KL divergence is the expectation of the log difference between the probability of data in the original distribution with the approximating distribution.

$$D_{KL}(p||q) = E[log\, p(x) - log\, q(x)] \tag{2.1}$$

Since $log\, a - log\, b = log\frac{a}{b}$ we can rewrite the equation as:

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot log\frac{p(x_i)}{q(x_i)} \tag{2.2}$$

where D_{KL} is The Kullback-Leibler divergence, p is the original distribution and q our approximating distribution.

2.3.3 Conditional variational autoencoder

So far, we have a generative model able to generate random samples from the latent space that resemble the input. The decoder cannot, however, produce a sample based on a particular condition on demand. Enter the conditional variational autoencoder (CVAE). This architecture has an extra input to both the encoder and the decoder. This additional input can be, for example, the label of the image you want to decode, or the properties of the molecule you are trying to reconstruct (which is our case).

By doing this, the neural network stops learning the properties of the condition we are already providing as second input, and focuses on other properties of the main input. This allows us not only to sample new data from the latent space, but also to specify the properties we want to have. For example, in the case of the MNIST dataset (in which the inputs are images of hand written numbers) the condition would be the

[3]Source: https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained

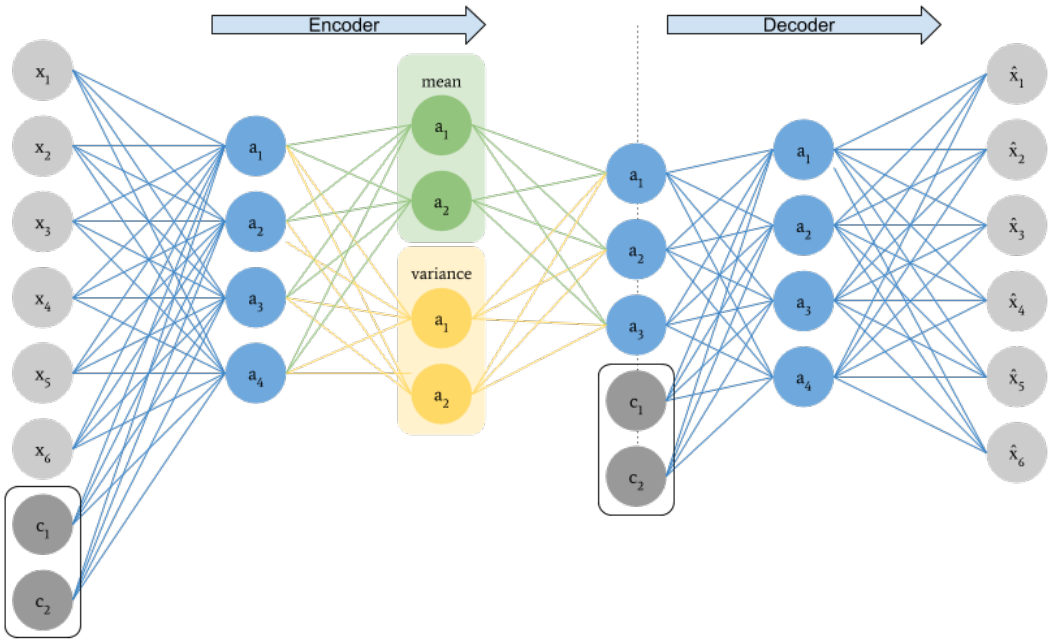


FIGURE 2.4: Structure of a conditional variational autoencoder (CVAE)

number itself one-hot encoded. Once the model is trained we can generate images of a specific number, by just random sampling from the latent space and concatenating it to the one-hot vector of the number we want.

The autoencoder is composed by two main parts: the encoder and the decoder. In our case, the encoder is a convolutional neural network (CNN) and the decoder a recurrent neural network (RNN).

2.4 Convolutional neural networks

CNNs are a category of neural networks that have been proven very effective in areas such as image recognition and classification [11, 12], but, since they are able to detect patterns between adjacent parts of the input, they are also useful when working with text [13].

A CNN is composed by one or more convolutional layers. Each layer has many feature detectors (also known as kernels), small data structures of 1, 2 or 3 dimensions depending on the problem. The output of each of these layers is the result of the dot product between the kernel and each patch of the input (as shown in Figure 2.5). Convolutional layers have a small amount of parameters compared with the fully-connected ones, but perform more operations.

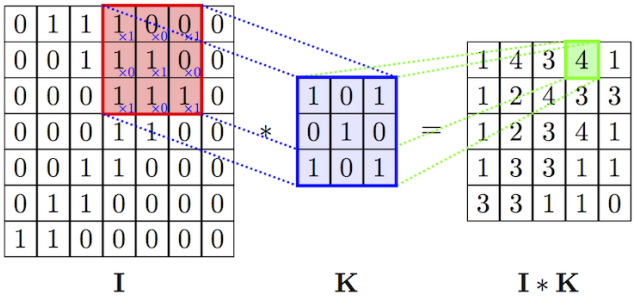


FIGURE 2.5: Example of a dot product operation between a kernel and a patch of an image (or any 2-dimensional input). Source: Deep Learning – Computer Vision and Convolutional Neural Networks

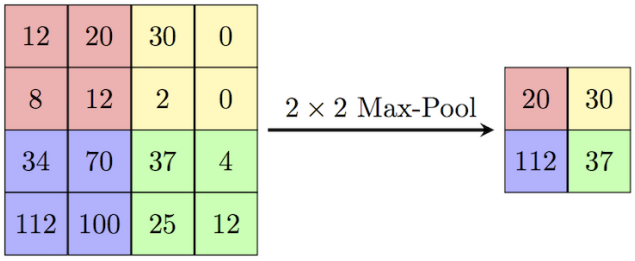


FIGURE 2.6: Example of a Max Pooling layer of size 2x2 applied to a 4x4 input. Source: ComputerScienceWiki

Mathematically, and for a 2-dimensional case, given a convolution kernel K represented by a $(M \times N)$ array, the convolution of an image I with K is:

$$output(x, y) = (I \otimes K)(x, y) = \sum_{m=0}^{M-1} \sum_{n=1}^{N-1} K(m, n) I(x - n, y - m) \tag{2.3}$$

It is very common, after some convolutional layers, to reduce the size of the output, either by adding pooling layers or by adding a stride in the convolution operation. A pooling layer of $(M \times N)$ outputs a single value for every window of $(M \times N)$ in the input (see Figure 2.6), and adding a stride of N in the convolution makes the kernel k to jump N positions instead of 1 while doing the convolution.

By doing this, the kernels of the convolutional layers in subsequent steps will extract different features, usually more generic or related to the whole input instead of just a little section. Another benefit is that the representation becomes more invariant to small transitions of the input, this is, that if we translate the input by a small amount, the values of most of the pooled outputs do not change.

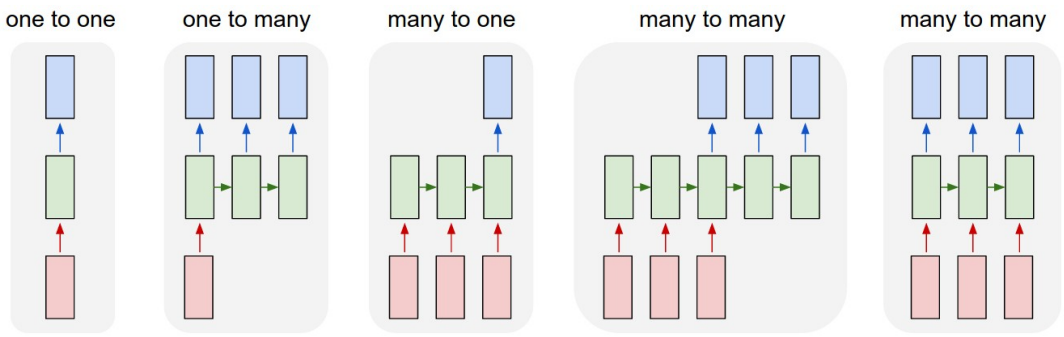


FIGURE 2.7: Each rectangle is a vector and arrows represent functions (e.g. dot product). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state

2.5 Recurrent neural networks

Classical neural networks, including convolutional ones, suffer from two severe limitations: They only accept a fixed-sized vector as input and produce a fixed-sized vector as output, and they do not consider the sequential nature of some data.

RNNs overcome these limitations by allowing to operate over sequences of vectors in the input, in the output, or both (Figure 2.7). They have proved to be very effective in the field of language [14], video frames [15] and time series [16], among others.

The basic formulas for a RNN are:

$$s_t = tanh(Ux_t + Ws_{t-1}) \tag{2.4}$$

$$y_t = Vs_t \tag{2.5}$$

These equations basically say that the current network state s_t commonly known as hidden state, is a function of the previous hidden state s_{t-1} and the current input x_t. U, V, W matrices are the parameters of the RNN and y_t is its output at time t. The visual concept is shown in Figure 2.8.

Given an input sequence, we apply RNN formulas in a recurrent way until we process all the input elements. This is what is called "unrolling in time of a RNN". The RNN shares the parameters U, V, W across all recurrent steps. Some important observations:

- We can think of the hidden state as a memory of the network that captures information about the previous steps. It embeds the representation of the sequence.

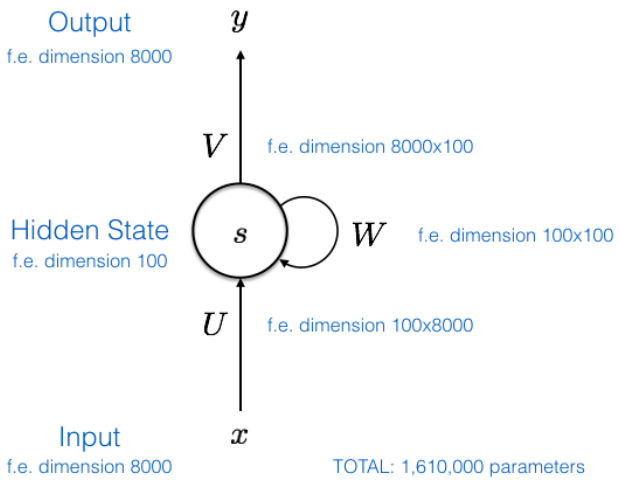


FIGURE 2.8: Representation of a RNN unit

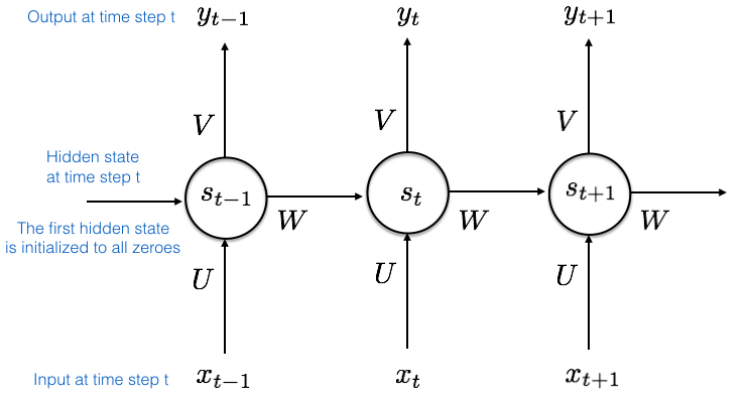


FIGURE 2.9: Complete sequence after unrolling many RNN units

- The output of the network can be considered at every stage (many to many) or only at the final one (many to one) as shown in Figure 2.7.
- When starting to train a RNN we must provide initial values for U, V, W as well as for s.

The output y is the probability distribution over the vocabulary (this is, all possible values) at each time-step t.

Training a RNN is similar to training a traditional NN, but with some modifications. The main reason is that parameters are shared by all time steps: in order to compute the gradient at $t = 4$, we need to propagate 3 steps and sum up the gradients. This is called backpropagation through time (BPTT).

Recurrent neural networks propagate weight matrices from one time-step to the next. Recall the goal of a RNN implementation is to enable propagating context

information through faraway time-steps. When this propagation results in a long series of matrix multiplications, weights can vanish or explode. We can mitigate these problems by using gradient clipping [17] or initialising the weights with random orthogonal matrices.

RNNs work just fine when we are dealing with short-term dependencies. However, vanilla RNNs fail to understand the long-term context dependencies (when relevant information may be separated from the point where it is needed by a huge load of irrelevant data). Gated RNNs (with units that are designed to forget and to update relevant information) are a solution to this problem.

There are two main types of gated RNNs: Long Short Term Memories (LTSM) [18] and Gated Recurrent Units (GRU) [19]. In our project, we will use GRUs.

2.5.1 Gated recurrent units

Gated recurrent units are designed in a manner to have more persistent memory thereby making it easier for RNNs to capture long-term dependencies. Let us define some new elements and how a GRU uses h_{t-1} and x to generate the next hidden state h_t.

$$Update\,gate : z_t = \sigma(W_z \cdot [x_t, h_{t-1}]) \tag{2.6}$$

$$Reset\,gate : r_t = \sigma(W_r \cdot [x_t, h_{t-1}]) \tag{2.7}$$

$$New\,memory : \tilde{h}_t = tanh(r_t \cdot [x_t, r_t \cdot h_{t-1}]) \tag{2.8}$$

$$Hidden\,state : h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t \tag{2.9}$$

It combines the forget and input gates into a single “update gate”. It also merges the cell state and hidden state, and makes some other changes. In general, GRUs are simpler than LSTMs, faster and optimise quicker.

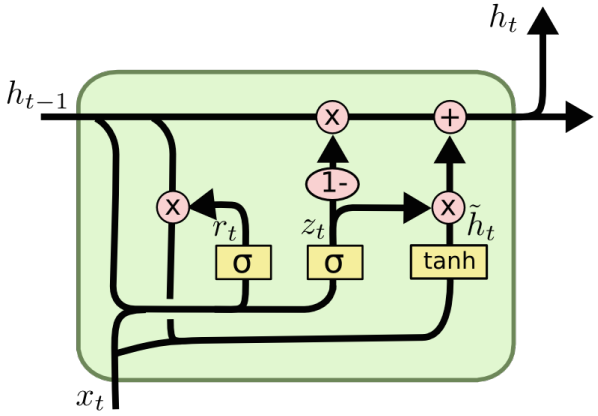


FIGURE 2.10: Architecture of a GRU unit

Chapter 3

State of the art

The field of de novo molecule generation has been widely studied [4, 5, 6, 7, 8, 9]. While we have the CC signatures to use as properties of the molecules, other researchers have used different properties, such as logP or cell-based transcriptional assays [20].

3.1 Deep Autoencoder RNNs with Generative Topographic Mapping

Some studies can be found in the field of de novo molecule generation. In their work, Sattarov et al. [9] did an autoencoder with Generative Topographic Mapping (GTM). From their study it can be highlighted that not only they mention variational autoencoders, but also provide the reasons why to not use them for molecule generation. The reason they provide is that the "property-molecule" relation is "one-to-many" (this is, the same property can be possessed by very similar molecules), which results in multimodal distributions of the molecules possessing the desired properties. On the other hand, a VAE approximates any point distribution by a single gaussian function, which implies the unimodality of that distribution and creates a contradiction with the first statement.

GTM is a technique for dimensionality reduction and data visualisation based on variational Bayesian statistics [21] [22]. As they claim in Sattarov et al. [9], GTM can be successfully used to approximate multimodal distributions, which is very important for performing de novo molecular design.

The architecture of they implement is shown in Figure 3.1. As we can see, the encoder consists of two bidirectional LSTM layers and a Dense layer that has as input the concatenation of the final cell states and hidden states from both LSTM. The

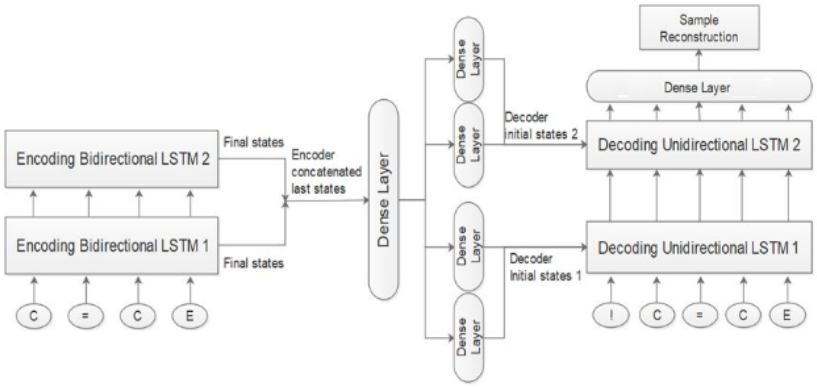


FIGURE 3.1: Architecture of the model developed by Sattarov et al. [9]

decoder, on the other hand, is composed by 4 parallel Dense layers that form the initial cell and hidden states for each unidirectional LSTM layer of the decoder.

3.2 Generative Adversarial Autoencoder

Nowadays, almost all the deep learning generative models are VAEs or generative adversarial networks (GAN). GANs are a type of generative neural network that differs with the variational autoencoder in the fact that, while the former generates samples by creating a representation space that follows a gaussian distribution, the latter trains a separate network that discriminates between the generated samples. In other words, GANs have a generator, and a discriminator.

In Kadurin el at. [8], they have implemented an adversarial autoencoder (AAE). It is a derivation of the GAN and, as it is shown in Figure 3.2, it has many similarities with a variational autoencoder. We can also see that the AAE has an additional loss compared to the variational autoencoder: the discriminator loss. This loss is distinguishing between latent variables and the standard gaussian.

As for the results, they claim 3 advantages of the AAE compared to the VAE:

- Adjustability in generating molecular fingerprints.
- Capacity of processing very large molecular datasets.
- Efficiency in unsupervised pretraining for regression model.

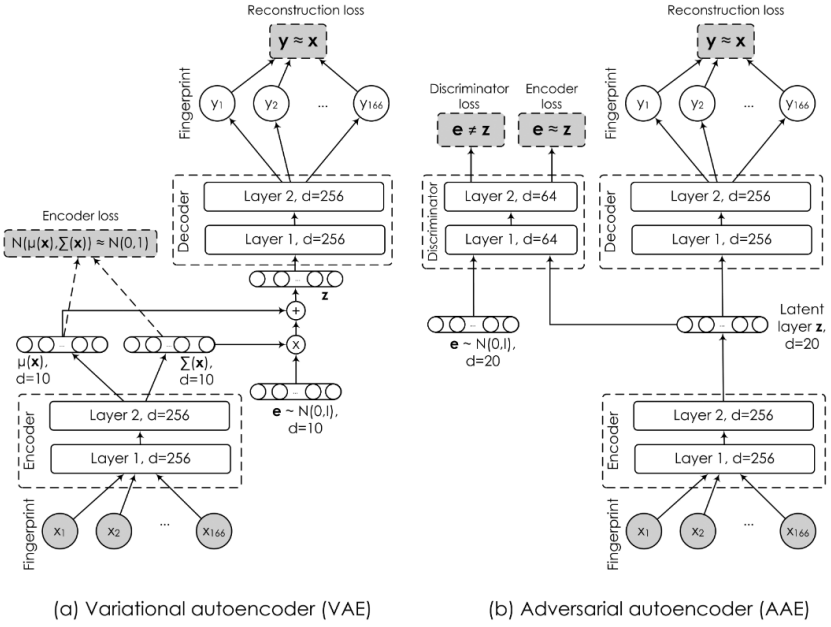


FIGURE 3.2: Comparison between the architecture of a VAE and an AAE, from Kadurin el at. [8]

Chapter 4

Methodology

4.1 Software

In order to implement the VAE and the CVAE, we considered two options: Tensorflow[1] and Keras[2].

The first one is an open source software library for numerical computation using data flows graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data array (tensors) communicated between them[3].

The second one is a framework built on top of Tensorflow that provides an abstraction for many tasks and makes the implementation of neural networks much simpler. On the other hand, it does not provide as much control as Tensorflow.

Tensorflow, on the other hand, has recently released the 2.0 beta version (July 2019) that includes some Keras functionalities, such as the layers and optimsers. Since we were already very confident with Keras, we wanted to use it instead of trying a new framework in which we were not as confident. In future projects, we will consider using pure Tensorflow as main framework.

4.2 Molecular representation & data preprocessing

As we already know, we are going to use a String representation of a molecule (SMILES) as input data. But we need to transform this string into something that the network can understand, this is, real numbers. In order to do so, we first map every possible character that a SMILES can have into a number. Afterwards, a padding of white spaces is applied to the right of the string until the string is 120 characters

[1] https://www.tensorflow.org/
[2] https://keras.io/
[3] Source: ethangoan's github.

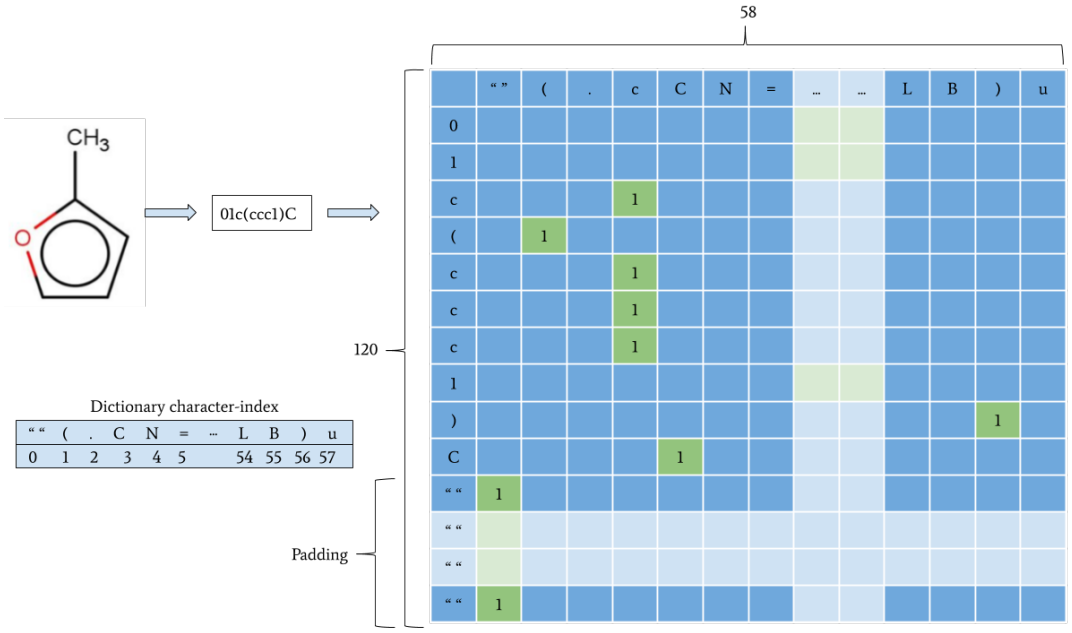


FIGURE 4.1: Example of the one-hot encoding technique applied to an example SMILES. As we can see, the SMILES is filled up with white spaces until it is 120 characters long, and then each of them is converted to an array of size 58, with all zeros except a one in the position specified by the "character-index" dictionary.

long. Once the size of the string is normalised we use the one-hot encoding technique, this is, we change every character of the SMILES for its corresponding number in the dictionary we previously stored and then we turn every number into a sparse vector of size 58, with all zeros except a one in the position of the corresponding number (see Figure 4.1).

4.3 HPC

In order to train the models, we got access to a cluster in the Barcelona Supercomputing Center (BSC). More concretely, to the CTE-POWER servers, that has clusters with 4 Tesla V100 GPUs and 160 CPUs. The specifications for each computer node are as follows[4]:

- 2 x IBM Power9 8335-GTH @ 2.4GHz (3.0GHz on turbo, 20 cores and 4 threads/core, total 160 threads per node)
- 512GB of main memory distributed in 16 dimms x 32GB @ 2666MHz
- 2 x SSD 1.9TB as local storage

[4]Source: BSC

- 2 x 3.2TB NVME
- 4 x GPU NVIDIA V100 (Volta) with 16GB HBM2.
- Single Port Mellanox EDR
- GPFS via one fiber link 10 GBit

The clusters provide the necessary modules to cover almost all the needs. In our case, we only needed to load a few modules, like the *tensorflow* and the *rdkit* ones.

Since they force you to use at least 40 CPUs for each GPU you use (to avoid bottlenecks), we implemented the preprocessing using the python's multithreading package. By doing this, we take advantage of the multiple cores and we cut the preprocessing time by a factor of 35.

We trained in parallel several models using 40 CPUs and one GPU. To send jobs to the cluster, we used the Slurm Workload Manager. According to the Slurm documentation[5], Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. Slurm requires no kernel modifications for its operation and is relatively self-contained.

By using the SBATCH directive in an executable file, it is possible to specify many options for the job, like its name, the number of CPUs to use, the number of GPUs, or the number of processes to run in parallel (for instance, running the same script x times)

[5]Source: Slurm documentation

Chapter 5

Development and results

5.1 First approach: VAE

The first model we used was a variational autoencoder we found in an internet article[1]. It consists of three 1-D convolutional layers as an encoder and 3 GRUs followed by a TimeDistributed layer, as we can see in Figure 5.1. The author trained the model with the ChEMBL database[2], which is a manually curated chemical database of bioactive molecules with drug-like properties. More specifically, he used a 80/20 train/test split, with 99% of the database, since he excluded molecules that have a SMILES larger than 120 characters.

The main purpose of this model is to learn the grammar of the SMILES. This means, to be able to represent the SMILES in a low-dimensional vector (latent space) to reconstruct it later.

The weights of the trained model were provided by the author, so no training was required in this approach.

The experiments done with this model consist of two main parts:

- First, we pass a set of molecules of size m through the encoder to get their latent representations.
- Second, since the latent space follows a gaussian distribution, we can sample using the latent space as the mean and adding a small noise (standard deviation). We generate n samples from each latent vector.

Since we are encoding m molecules, and trying to reconstruct each one n times, we are generating $m \times n$ potential new molecules. Also, we perform these reconstructions

[1] Source: Using autoencoders for molecule generation
[2] https://www.ebi.ac.uk/chembl/

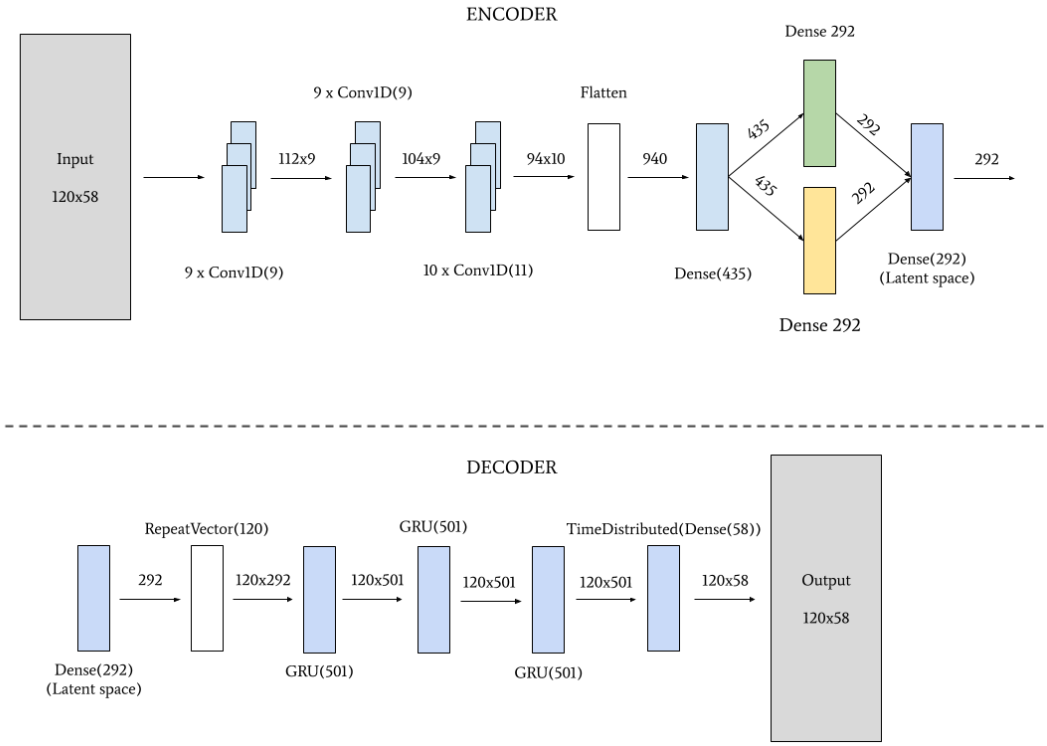


FIGURE 5.1: Architecture of the grammarVAE model

for 3 different noises applied to the latent space, more specifically, 0, 0.05 and 0.1 standard deviations.

The molecules used for the prediction were extracted from the DrugBank database[3]. Since the experiment takes time, and we wanted this step of the project to be fast, we split the experiment in two parts: Instead of reconstructing 1,000 molecules (m) 1,000 times (n), we did the experiment for $m = 1000$, $n = 100$ and $m = 100$, $n = 1000$. With the first experiment we focused on the behaviour of the reconstructions with respect to the input molecules, and with the second one we focused on the behaviour of the reconstructions itself. For the sake of simplicity, and since the data we use is from the same dataset, we will show the plots from both experiments as if it was only one, using the plots that are more convenient for each situation.

5.1.1 Results

For this experiment, most of the SMILES were between 20 and 80 characters long (Figure 5.2), and the most common characters are 'C' and 'c', which correspond to the carbon atom, being the second one a special case for aromatic rings (see https://en.wikipedia.org/wiki/Aromaticity) (Figure 5.3).

[3] https://www.drugbank.ca/

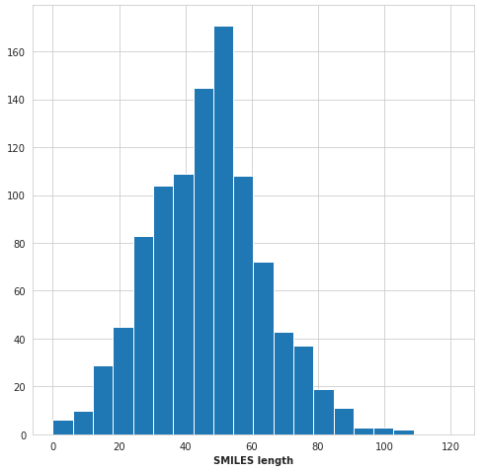


FIGURE 5.2: Distribution of SMILES length

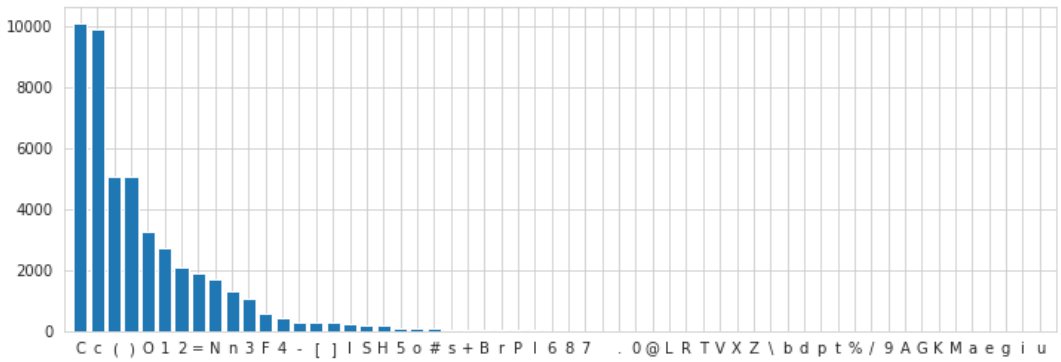
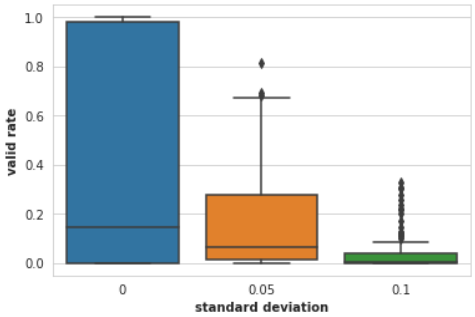
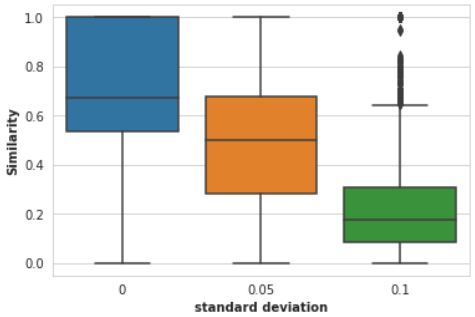


FIGURE 5.3: Distribution of SMILES characters



(A) Ratio of valid molecules



(B) Tanimoto similarity of the reconstructions

FIGURE 5.4: On the left, the ratio of reconstructed molecules that are valid. On the right, the Tanimoto similarity between the reconstructed valid molecules and their original molecule.

In Figure 5.4a we can see the ratio of valid reconstructions. It is very sparse in the case of 0 noise, but as long as the standard deviation is increased, we can see that the ratio goes down, meaning that most of the molecules we are generating are not valid molecules. On the other hand, Figure 5.4b shows that the Tanimoto similarity[4] for valid reconstructions also decreases as the standard deviation is increased.

Furthermore, as shown in Figure 5.5, some characters in the input SMILES are more likely to be correctly reconstructed than others. There is also a clear pattern between the size of the input length and the ratio of valid reconstructions. As shown in Figure 5.6, the model performs better when it comes to short SMILES, which makes sense, since big molecules are more complex and for instance more difficult to be represented in the latent space without any loss.

So far, it looks like the best option is to not add noise to the reconstructions. But there is an important fact we are missing here: we are not checking how many reconstructions are unique. If we have a 100% accuracy in the molecule reconstruction, but all of them are equal, in the end we are failing in two things: The amount of different molecules we are generating is less, and the diversity is null. The main goal is to generate new molecules that are somehow different to the input ones, so having 0 diversity does not help at all.

In Figure 5.7a we can see that the percentage of unique molecules in the 0-noise case is really low (around 1.5%), but by adding noise with a standard deviation of 0.05 we get around 12% of unique valid molecules. In Figure 5.7b we removed the "unique valid molecules" bar, so it is easier to see the other bars. The first one represents the

[4] Tanimoto similarity

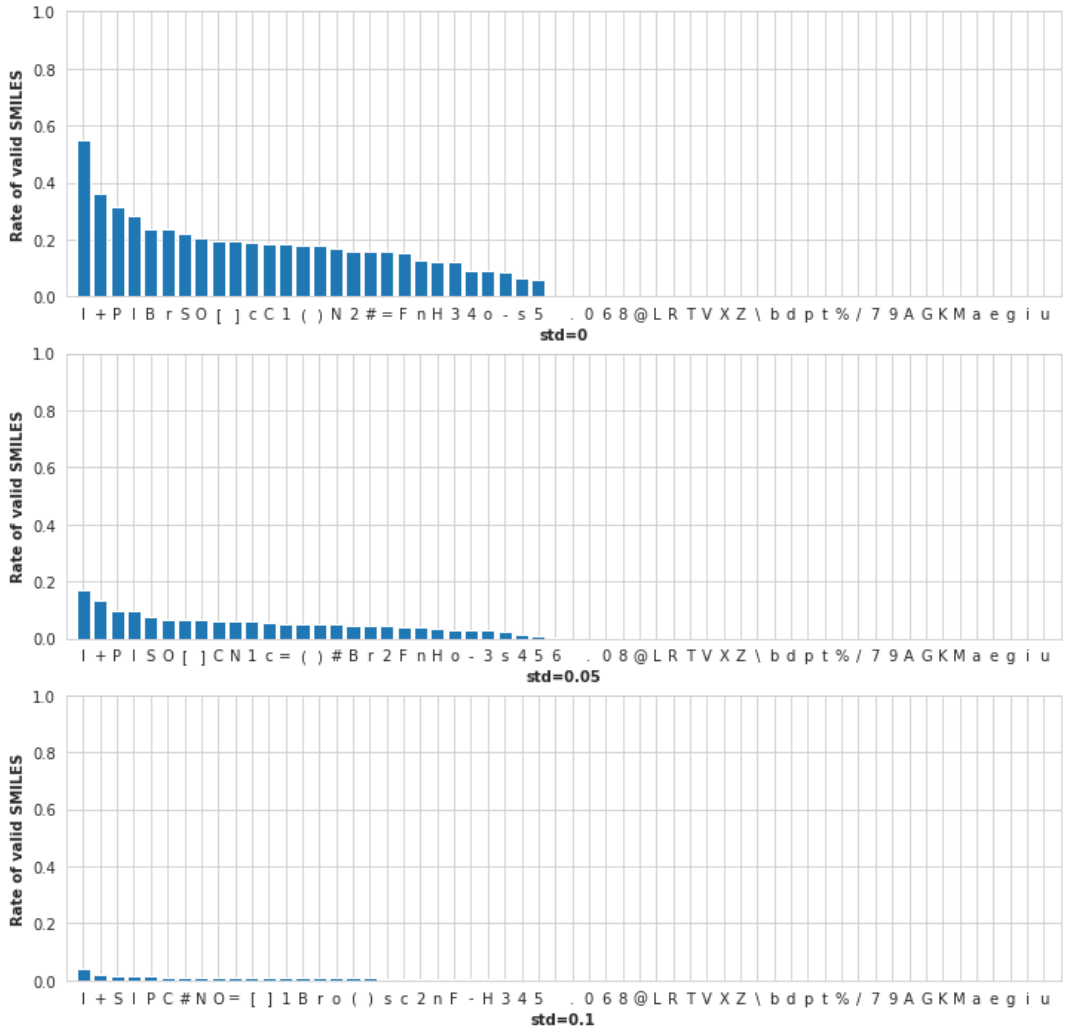


FIGURE 5.5: Valid reconstructions for every character

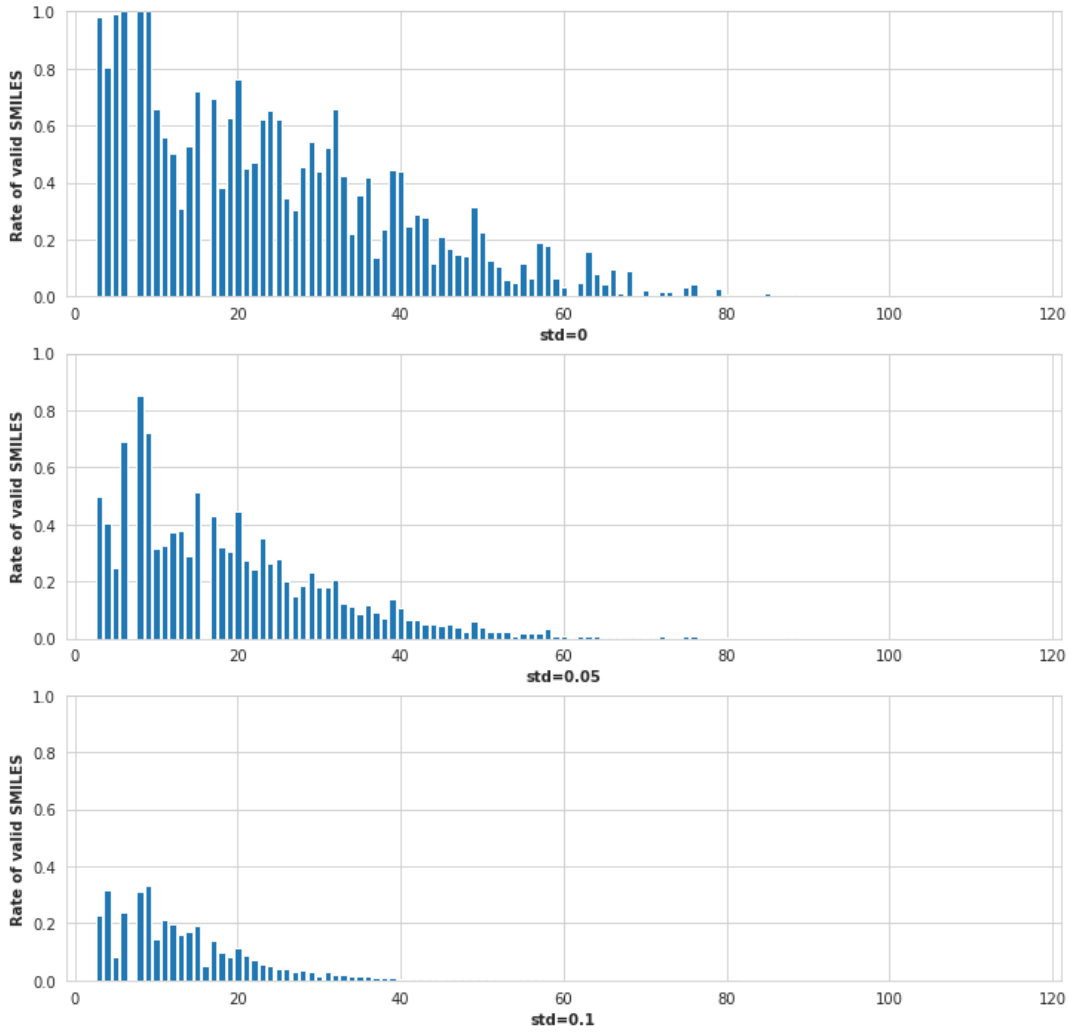
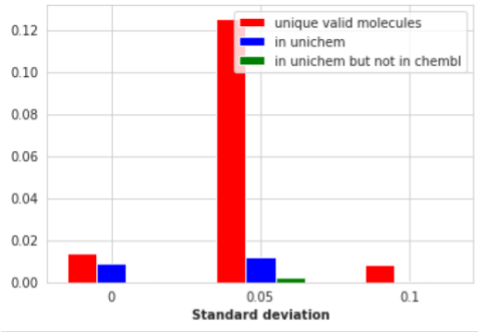
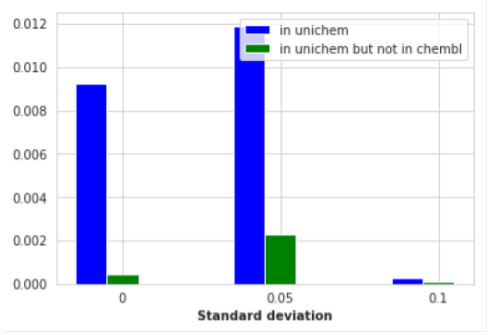


FIGURE 5.6: Valid reconstructions based on SMILES length



(A) Length variation for valid reconstructions



(B) Length variation for valid reconstructions

amount of molecules that are in UniChem[5]. This means that those molecules that we generated are molecules that are registered in a database, which means that they are valid and/or useful. In order to check that these molecules are not due to overfitting, we check how many of these molecules are not in ChEMBL, the database that was used to train the model.

5.2 Second approach: CVAE

In this approach we implemented a conditional variational autoencoder. This means that this time we will be able to provide additional information to the autoencoder, in particular the properties of the molecule, which is a vector of 128 dimensions. As we can see in Figure 5.8, the signature is repeated and concatenated vertically to the molecule input (one-hot encoded matrix of dimensions 120x58). After the encoder is applied and we get the latent space, we concatenate the same signature again into that latent space. Then, we apply the decoder normally.

To train the model, we were provided with 500k molecules from the laboratory's database, with their corresponding signatures (more specifically, signature type 3 B4). The model was trained with 400k molecules in a high performance cluster (details in section 4.3), using 20% of the data as validation data. After a few hundred epochs, we got a validation accuracy of 99.5%.

The model was trained applying the categorical cross-entropy loss function at the reconstruction, and the Kullback-Leibler divergence at the latent space. The learning rate used was 10^{-4}. We also tried a value of 10^{-5}, but the only difference was the speed of convergence (the first one converged faster). Since all fine-tuning modifications were

[5] ebi.ac.uk/unichem

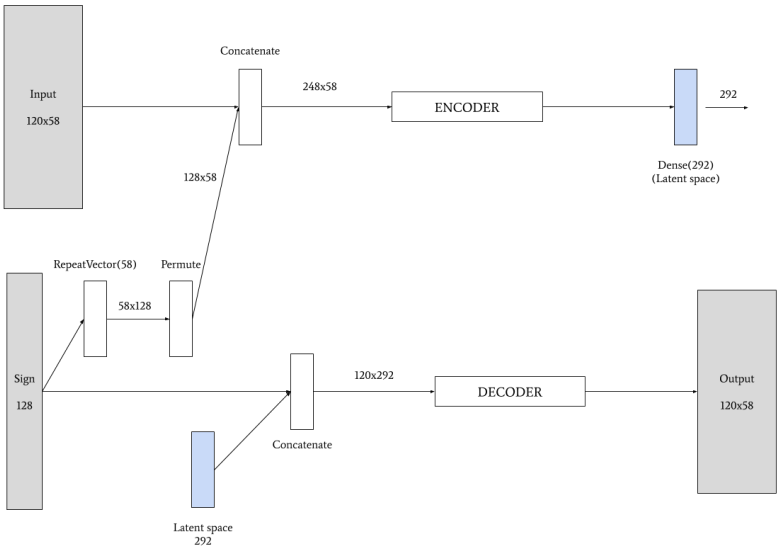


FIGURE 5.8: Architecture of the customised conditional variational autoencoder. The encoder and decoder have the same architecture as in the VAE. The only difference remains in the input. Note that the latent space (blue rectangle) is the same for both the output of the encoder and the input of the encoder.

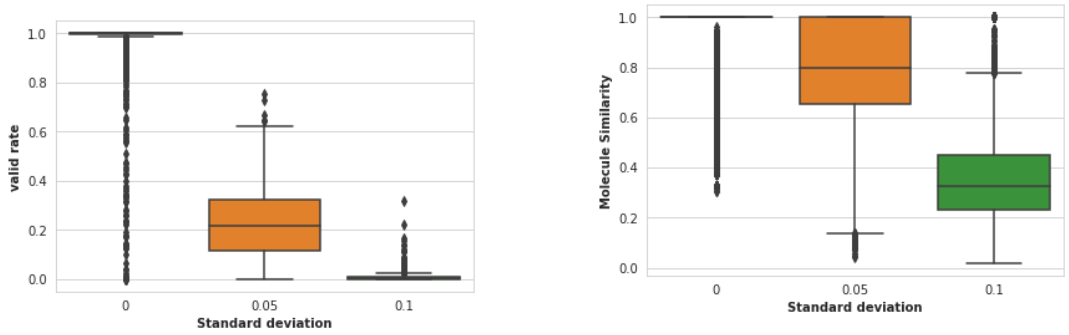
achieving an accuracy of 99.5%, we used the default learning rate, which achieved the convergence faster (10^{-4}).

In the experiments done with this model, we will show some plots that are the same as the ones made for the VAE, but we will omit other plots, like length and character distribution, or how the reconstructions behave in terms of length or validity for every character. Instead, we will show some new plots, specially in the second experiment.

5.2.1 First experiment: Fix the signature

In the first experiment, we got a set of 1,000 molecules that were not part of the training, and encoded them with their respective signatures in order to get their latent space. Once we had 1000 latent spaces, we added noise with different standard deviations, exactly in the same way we did in the experiments with the VAE, generating 1,000 copies of each latent space. After that, we concatenated each latent space with their respective signatures again, and got the reconstructed molecules (1 million in total).

As we can see in Figure 5.9a, the standard deviation is highly affecting the ratio of valid molecules we get after decoding. Almost all molecules generated with $std = 0$ are valid molecules, and almost any molecule generated with $std = 0.1$ is valid. As shown in Figure 5.9b, the Tanimoto similarity between the generated molecules and



(A) Ratio of valid molecules

(B) Tanimoto similarity of the reconstructions

FIGURE 5.9: On the left, the ratio of reconstructed molecules that are valid. On the right, the Tanimoto similarity between the reconstructed valid molecules and their input molecule.

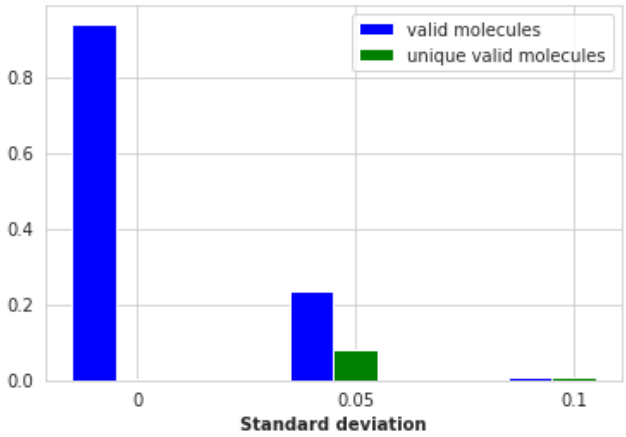


FIGURE 5.10: Proportion of valid and unique molecules for the first CVAE experiment.

their respective input is also heavily affected by the standard deviation, being the case of $std = 0$ not a good choice, since what we want is to generate new molecules that are slightly different, but not as different as using $std = 0.1$. Moreover, as shown in Figure 5.10, the optimal standard deviation to apply to the latent space in order to maximise the number of unique valid molecules is $std = 0.05$.

Taking all this considerations, we conclude that the optimal standard deviation in order to generate a decent amount of somehow similar molecules is 0.05. We will use this value to apply noise to all latent spaces in the next experiment.

5.2.2 Second experiment: Sample different signatures

This experiment differs from the last one in the sense that instead of just applying noise to the latent space and generate new molecules (which is what we have been

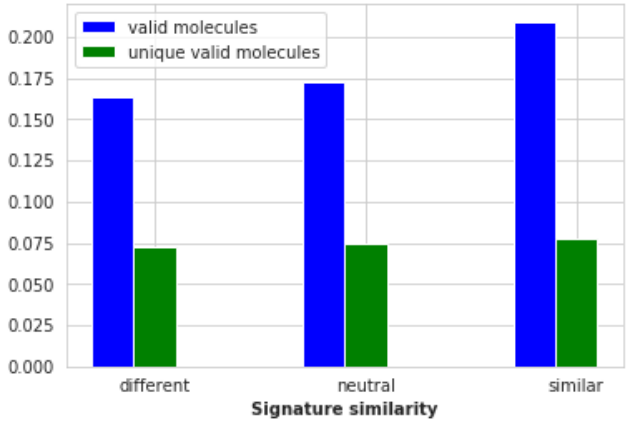


FIGURE 5.11: Proportion of valid and unique reconstructed molecules for the second CVAE experiment.

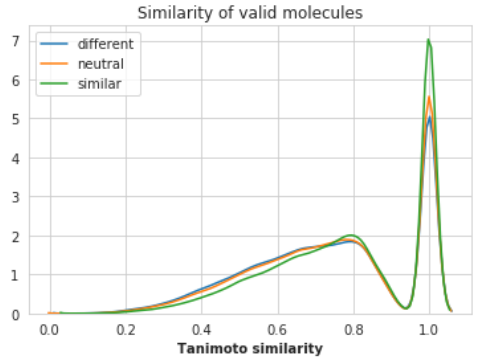
doing until now), we will apply a fixed noise of $std = 0.05$ (which showed a better performance), and then we will sample different signatures that will be appended to the latent space. The motivation of this experiment is that we want to check if, after the molecules are decoded, they preserve the properties of the signatures sampled from the dataset. If we manage to do so, it means that we are able to generate new molecules from a molecule M_0 with properties S_0, that accomplishes different target properties from the sampled signatures $S_1, S_2, ..., S_n$.

For the experiment, we will get a set of 1,000 molecules that were not part of the training set, and encode them with their respective signatures to get their latent space. Once we have the latent spaces, we sample for each one a set of 1,000 signatures that are very similar to the original one, 1,000 signatures that are very different, and 1,000 signatures that are neither similar or different (we called them neutral). In order to do this sampling, we compute the cosine similarity between the signature of the input molecule and all the signatures in the dataset.

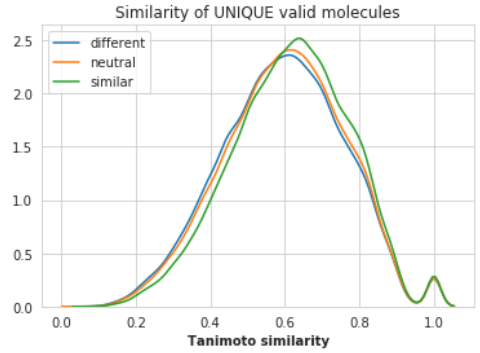
One we have the 3,000 signatures (1,000 of each group) we proceed to perform the decoding from the latent space (plus the new sampled signatures) to the new molecules.

First, let's analyse the ratio of unique molecules that we have generated. As we can see in Figure 5.11, the total amount of unique molecules does not increase that much if we use similar signatures at decoding time. This can mean that the signature is not affecting very much the prediction, fact that will be denied in posterior plots (Figure 5.14b).

In Figure 5.12a we can see the distribution of the similarities of the generated



(A) Tanimoto similarity of ALL the reconstructions



(B) Tanimoto similarity of the reconstructions of VALID molecules

FIGURE 5.12: On the left, the Tanimoto similarity between the reconstructed valid molecules and their respective input. On the right, the same comparison but only taking into account unique valid molecules, this is, removing the repeated reconstructions.

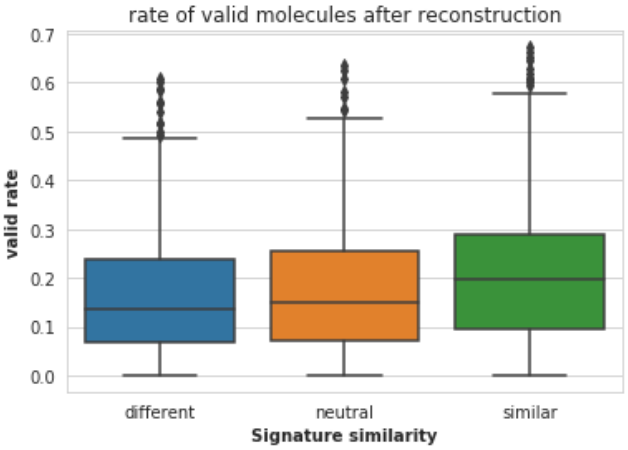
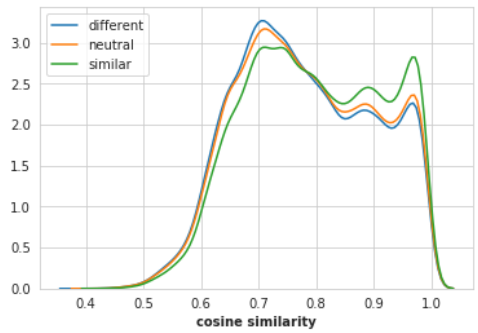


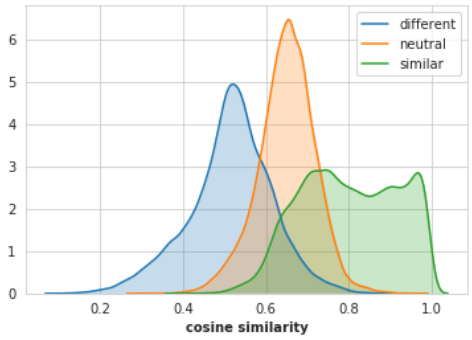
FIGURE 5.13: Ratio of valid reconstructions based on 3 different levels of signature similarity.

molecules with respect to the input ones. The peak at similarity 1 is because all those generated molecules are basically the same as the input. This is due to the nature of a regular autoencoder: they try to reconstruct the input with the maximum accuracy possible. After removing the duplicated molecules, we plotted a second distribution of the similarities taking into account the unique set of molecules (Figure 5.12b). It does not show a clear difference between the reconstructions from signatures similar or different to the input molecule.

Regarding the ratio of valid reconstructions (Figure 5.13), we can see again that the type of signature chosen for the reconstruction does not affect very much the ratio. There is a slight increase in the ratio for the reconstructions where signatures similar to the input molecule were used, but it is not very significant.



(A) Cosine similarity between signatures of predicted molecules and input signatures



(B) Cosine similarity between signatures of predicted molecules and the signatures sampled for the decoding.

FIGURE 5.14

Up to this point, we saw that the signatures used in the decoding are not affecting very much the result, but there is still something we have to check. We have to check if the signatures used at decoding time are affecting the signatures of the predicted molecules. In order to check this, we compute the signatures of the predicted molecules using the Chemical Checker module, and we compare it with the signature of the molecule that was used as input (Figure 5.14a) and with the signatures that were sampled at decoding time (Figure 5.14b).

As we can see in Figure 5.14a, similarity between the predicted signatures of the reconstructed molecules and the signature of their corresponding input molecule does not change based on the type of signature chosen for the decoding.

On the other hand, it does affect when we compare those signatures of predicted molecules with the corresponding signature used for the decoding (Figure 5.14b). A perfect plot here would show all the distributions having all the area as close as possible to 1. This would mean that, no matter the type of signature used for decoding, the signature of the generated molecule would resemble the signature used to create that molecule. It makes sense though, that signatures that are very different from the original one will create a conflict in the network and the generated molecule will not have a very similar signature. This is due to the fact that when we train the network, the signature provided to the input and to the latent space resembles the properties of that molecule, and we are now changing the signature we add to the latent space for properties that have nothing to do with the molecule. Basically, what Figure 5.14b is saying, is that it is harder to generate new molecules with properties very different to the original one (yet possible). On the other hand, using very similar signatures give

us new molecules that are probably not very similar to the original one (see Tanimoto similarity in Figure 5.12b), but with very similar properties.

Chapter 6

Limitations and possible improvements

There are two main factors that limited the growth of this project: Time and field knowledge. The lack of time made us go straight to the point and and prevented us from trying other procedures. Regarding field knowledge, the lack of a specialised background in biology/chemistry meant investing more time in learning basic concepts, and also limited our perspective of the global problem and hence our ability to propose new ideas that are only possible with that base knowledge.

Ideas for future studies and implementations that can be improved have been thought:

- First, when one-hot encoding the SMILES, something we realised is that some atomic symbols are composed by two characters (i.e. Cl for Chlorine). Despite the network should be able to recognise these patterns (due to the nature of CNNs and RNNs), we think it could be a good idea to encode those atomic symbols as if they were only one character.

- For the last experiment, we could check if the generated molecules that have the properties of interest belong to some well-known molecule database. By doing this, we could know if the generated molecules are actually useful or not.

- Another suggestion is to check that the properties of the generated molecules are closer to the target signature than the signature of the original input. By doing this, we could measure how the original molecule affects to the inference of new properties.

- We had an ambitious idea which we already knew it was going to be hard to perform. GANs are a well known generative network [4] [5] [7] [8], but they are

extremely difficult to train[1]. For this reason, we had them at the very end of our to-do list, but we also think that it is worth to take a look and try to train one.

[1] GAN — Why it is so hard to train Generative Adversarial Networks!

Chapter 7

Conclusion

This Master's Thesis was composed of two main parts: The development of a VAE, and the development of a CVAE. In the first one, we analysed the state-of-the art regarding to de novo molecule generation using VAEs and we implemented an already-existing model[1] as a first approach. Since the model was already trained, we focused on exploring the results obtained with new molecules and learned how an autoencoder works.

Many tests were made with this first model. We tested how the noise in the latent space affects the quality of the generated molecules, and used this information in the last experiment of the second approach.

With the second model, the CVAE, we modified the architecture of the previous VAE and added the CC signature of the molecule to the model. In order to do this, we looked at simple examples of CVAEs and adapted them to our current VAE[2][3].

After that, we trained the model in a High Performance Cluster with 400k molecules and their corresponding signatures until we got 99.5% accuracy in the reconstructions. Then, we performed two main tests: In the first one, we checked again how adding noise to the latent space affects the generated molecules, and in the second one we sampled target signatures of interest and changed them in the input of the decoder.

With the second experiment, we saw that our model was able to generate new molecules with signatures that were close to the target ones. These experiments open a new opportunity for research in future projects with the help of the signatures from the CC or any other embedding representation of the properties of a molecule.

[1] Source: Using autoencoders for molecule generation
[2] Conditional Variational Autoencoders
[3] Conditional generation via Bayesian optimisation in latent space

Bibliography

[1] Jérôme Hert, John J Irwin, Christian Laggner, Michael J Keiser, and Brian K Shoichet. Quantifying biogenic bias in screening libraries. *Nature chemical biology*, 5(7):479, 2009.

[2] Christopher M Dobson. Chemical space and biology. *Nature*, 432(7019):824, 2004.

[3] Miquel Duran-Frigola, Eduardo Pauls, Oriol Guitart-Pla, Martino Bertoni, Modesto Orozco-Ruiz, Victor Alcalde, Victor M Diaz, Antoni Berenguer-Llergo, David Amat, Teresa Juan-Blanco, et al. Extending the small molecule similarity principle to all levels of biology. *Available at SSRN 3380254*, 2019.

[4] Dongyu Xue, Yukang Gong, Zhaoyi Yang, Guohui Chuai, Sheng Qu, Aizong Shen, Jing Yu, and Qi Liu. Advances and challenges in deep generative models for de novo molecule generation. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 9(3):e1395, 2019.

[5] Benjamin Sanchez-Lengeling and Alán Aspuru-Guzik. Inverse molecular design using machine learning: Generative models for matter engineering. *Science*, 361(6400):360–365, 2018.

[6] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. Deep reinforcement learning for de novo drug design. *Science advances*, 4(7):eaap7885, 2018.

[7] Oscar Méndez-Lucio, Benoit Baillif, Djork-Arné Clevert, David Rouquié, and Joerg Wichard. De novo generation of hit-like molecules from gene expression signatures using artificial intelligence, 2018.

[8] Artur Kadurin, Sergey Nikolenko, Kuzma Khrabrov, Alex Aliper, and Alex Zhavoronkov. drugan: an advanced generative adversarial autoencoder model for de novo generation of new molecules with desired molecular properties in silico. *Molecular pharmaceutics*, 14(9):3098–3104, 2017.

[9] Boris Sattarov, Igor I Baskin, Dragos Horvath, Gilles Gérard Marcou, Esben Jannik Bjerrum, and Alexandre Varnek. De novo molecular design by combining deep autoencoder recurrent neural networks with generative topographic mapping. *Journal of chemical information and modeling*, 2019.

[10] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

[11] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5297–5307, 2016.

[12] Qing Li, Weidong Cai, Xiaogang Wang, Yun Zhou, David Dagan Feng, and Mei Chen. Medical image classification with convolutional neural network. In *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, pages 844–848. IEEE, 2014.

[13] Cicero Dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, 2014.

[14] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[15] Lianli Gao, Zhao Guo, Hanwang Zhang, Xing Xu, and Heng Tao Shen. Video captioning with attention-based lstm and semantic consistency. *IEEE Transactions on Multimedia*, 19(9):2045–2055, 2017.

[16] Michael Hüsken and Peter Stagge. Recurrent neural networks for time series classification. *Neurocomputing*, 50:223–235, 2003.

[17] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2, 2012.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479, 1997.

[19] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[20] Xiao Yang and Cong Ma. In vitro transcription assays and their application in drug discovery. *JoVE (Journal of Visualized Experiments)*, (115):e54256, 2016.

[21] Christopher M Bishop, Markus Svensén, and Christopher KI Williams. Gtm: The generative topographic mapping. *Neural computation*, 10(1):215–234, 1998.

[22] Christopher M Bishop, Markus Svensén, and Christopher KI Williams. Developments of the generative topographic mapping. *Neurocomputing*, 21(1-3):203–224, 1998.